Computational Complexity and Numerical Stability of Linear Problems

Olga Holtz Noam Shomron

Jun 18, 2009

Abstract

We survey classical and recent developments in numerical linear algebra, focusing on two issues: computational complexity, or arithmetic costs, and numerical stability, or performance under roundoff error. We present a brief account of the algebraic complexity theory as well as the general error analysis for matrix multiplication and related problems. We emphasize the central role played by the matrix multiplication problem and discuss historical and modern approaches to its solution.

1 Computational complexity of linear problems

In algebraic complexity theory one is often interested in the number of arithmetic operations required to perform a given computation, modelled as a programme which receives an input (a finite set of elements of some algebra) and performs a sequence of algebra operations (addition, subtraction, multiplication, division, and scalar multiplication). This is called the total (arithmetic) complexity of the computation.[1] Moreover, it is often appropriate to count only multiplications (and divisions), but not additions or multiplications by fixed scalars. These notions can be formalized [BCS97, Definition 4.7]. For now, let us invoke

Notation. Let $\mathbb{F}$ be a field, let $\mathbb{F}[x_1,\dots,x_n] \subseteq A \subseteq \mathbb{F}(x_1,\dots,x_n)$ be an $\mathbb{F}$-algebra, and let $\Phi = \{\varphi_1,\dots,\varphi_m\}$ be a finite set of functions. The total arithmetic complexity of Φ will be denoted $L_A^{\mathrm{tot}}(\Phi)$, and its multiplicative complexity by $L_A(\Phi)$.

Intuitively, this is the minimal number of steps required to compute all of $\varphi_1, \dots, \varphi_m$ starting from a generic input $(x_1,\dots,x_n)$, with intermediate results in A (in all cases we consider, A will simply be the algebra of polynomials or of rational functions in the input variables, and will not always be explicitly indicated). The input and all intermediate results are understood to be stored in memory, and the simultaneous computation of a set Φ of functions means that at the end of the programme Φ is contained in the set of results.[2]

Let U, V, and W be finite-dimensional vector spaces over $\mathbb{F}$, and consider the class of bilinear functions $\varphi\colon U \times V \to W$ (which includes matrix multiplication). To define the multiplicative complexity of such a function, choose bases $\{u_i\}_{1\le i\le m}$, $\{v_j\}_{1\le j\le n}$, and $\{w_k\}_{1\le k\le p}$, so that

$$\varphi\left(\sum_{i=1}^{m} x_i u_i, \sum_{j=1}^{n} y_j v_j\right) = \sum_{k=1}^{p} \varphi_k w_k.$$

[1] This model only counts the operations and completely ignores storage and communications costs, limitations on precision, details of how arithmetic is implemented (in particular we are not counting *bit* operations), etc.

[2] For example, $X_0 := x,\ X_1 := X_0 \cdot X_0,\ X_2 := X_1 \cdot X_1,\ X_3 := X_2 \cdot X_0$ is a programme, using 3 operations—solely multiplications in this example, which computes x^5, as well as any subset of $\{x, x^2, x^4, x^5\}$, in $A = \mathbb{F}[x]$.

We regard the coefficients as variables, so that each φ_k is a homogeneous polynomial of degree 2 in $x_1, \ldots, x_m, y_1, \ldots, y_n$.

Definition (Cf. [BCS97, Definition 14.2]). $L(\varphi) = L_{\mathbb{F}[x_1,\ldots,x_m,y_1,\ldots,y_n]}\big(\{\varphi_1, \ldots, \varphi_p\}\big)$.

Because we are considering the multiplicative complexity, this is a well-defined notion that does not depend on the choice of bases.

It turns out that the multiplicative complexity of a bilinear function $\varphi\colon U \times V \to W$ is controlled by a somewhat more well-behaved notion, the *rank* $R(\varphi)$. This is a standard notion in multilinear algebra, which generalizes that of the rank of a linear map.

Definition. Let $t \in V_1 \otimes \cdots \otimes V_n$. The rank $R(t)$ is the smallest r such that one can write $t = \sum_{i=1}^{r} t_i$ with each t_i a monomial tensor, i.e., of the form $t_i = v_1 \otimes \cdots \otimes v_n$ for some $v_i \in V_i$.

In case $\varphi\colon U \times V \to \mathbb{F}$ is the bilinear map corresponding to a linear function $\widetilde{\varphi}\colon U \to V^*$, the rank $R(\varphi)$ is the rank of $\widetilde{\varphi}$ in the usual sense. Well-known algorithms, such as Gaussian elimination, as well as the fast algorithms described in this paper (see Section 1.4), can quickly compute the rank of a matrix, but determining the rank of a tensor of order 3 already seems to be quite difficult. Computing the rank of a given tensor is a combinatorial or algebro-geometric problem [Lan08].

We now explain how the rank controls the complexity of a bilinear function. First, by a known result of Strassen (see [BCS97, Proposition 14.4]), if $\varphi\colon V \to W$ is a *quadratic* map between finite-dimensional vector spaces, that is,

$$\varphi\left(\sum_{i=1}^{n} x_i v_i\right) = \sum_{j=1}^{p} \varphi_j(x_1, \ldots, x_n) w_j$$

for some bases $\{v_i\}_{1\le i\le n}$ (resp. $\{w_j\}_{1\le j\le p}$) of V (resp. W) and homogeneous polynomials $\varphi_i \in \mathbb{F}[x_1, \ldots, x_n]$ of degree two, then we need not search through some rather large class of programmes to find one which computes φ optimally, for in fact $L(\varphi) = L_{\mathbb{F}[x_1,\ldots,x_n]}\big(\{\varphi_1, \ldots, \varphi_p\}\big)$ equals the smallest $l \ge 1$ such that

$$\varphi(v) = \sum_{i=1}^{l} f_i(v) g_i(v) w_i \tag{1}$$

for some linear functionals f_i, $g_i \in V^*$. (Note that such a formula immediately gives an obvious algorithm computing $\varphi(v)$ using only l (non-scalar) multiplications.)

Now let $\varphi\colon U \times V \to W$ be a bilinear map between finite-dimensional vector spaces. This is covered by the preceding result of Strassen, since a bilinear map $U \times V \to W$ may be regarded as a quadratic map via the isomorphism $\mathbb{F}[U \times V] = \mathbb{F}[U] \otimes \mathbb{F}[V]$. A *bilinear algorithm* for φ amounts to writing

$$\varphi(u, v) = \sum_{i=1}^{r} f_i(u) g_i(v) w_i \tag{2}$$

for certain linear functionals $f_i \in U^*$, $g_i \in V^*$, and $w_i \in W$. The minimum such r is the rank $R(\varphi)$. Note that the rank of φ is not necessarily the same as its bilinear complexity, despite the superficially similar-looking formulae (1) and (2). However, by decomposing a linear functional $f\colon U \times V \to \mathbb{F}$ as $f(u, v) = f(u, 0) + f(0, v)$, one can see that

$$L(\varphi) \le R(\varphi) \le 2L(\varphi).$$

It is often easier to work with the rank rather than the more subtle notion of multiplicative (or total) complexity, and the above inequality shows we do not lose much in doing so.

1.1 Algebraic complexity of matrix multiplication

The basic problem is to compute the (total or multiplicative) complexity of multiplying two $n \times n$ matrices. This is a difficult question whose answer is not at present known for $n = 3$, for instance.

Matrix multiplication is a bilinear problem (see Section 1)

$$\begin{aligned} \varphi\colon M_{n\times n}(\mathbb{F}) \times M_{n\times n}(\mathbb{F}) &\to M_{n\times n}(\mathbb{F}) \\ (X, Y) &\mapsto XY = \left(\sum_{l=1}^{n} X_{il}Y_{lj}\right)_{1\le i,j\le n} \end{aligned}$$

whose corresponding tensor will be denoted

$$\langle n, n, n\rangle := \sum_{1\le i,j,k\le n} u_{ij} \otimes v_{jk} \otimes w_{ki}.$$

For $n = 2$ Winograd proved [Win71] that seven multiplications are required, so $L\big(\langle 2, 2, 2\rangle\big) = R\big(\langle 2, 2, 2\rangle\big) = 7$, but for $n = 3$ even the rank is not known at present (it is known that $19 \le R\big(\langle 3, 3, 3\rangle\big) \le 23$; see [BCS97, Exercise 15.3], [Lan08]).

Instead of fixing n, one considers the *asymptotic* complexity of matrix multiplication:

$$\omega(\mathbb{F}) = \inf\left\{ \tau \in \mathbb{R} \;\middle|\; L^{\mathrm{tot}}_{\mathbb{F}[X_{ij},Y_{ij}]}\left(\left\{ \sum_{l=1}^{n} X_{il}Y_{lj} \;\middle|\; 1 \le i, j \le n \right\}\right) = O(n^{\tau}) \right\} \tag{3}$$

so that $n \times n$ matrices with entries in $\mathbb{F}$ may be multiplied using $O(n^{\omega(\mathbb{F})+\eta})$ operations,[3] for every $\eta > 0$.

First of all, one can replace the total complexity in (3) by the multiplicative complexity or by the rank [BCS97, Proposition 15.1] and get the same exponent. Second, $\omega(\mathbb{F})$ is invariant under extension of scalars [BCS97, Proposition 15.18], so it does not depend on the exact choice of field $\mathbb{F}$ (e.g., $\mathbb{Q}$ versus $\mathbb{R}$ or $\mathbb{C}$), but rather only on its characteristic, which is usually taken to be zero (so ω denotes $\omega(\mathbb{C})$).

The value of ω is an important quantity in numerical linear algebra, as it determines the asymptotic complexity of not merely matrix multiplication but also matrix inversion, various matrix decompositions, evaluating determinants, etc. (see Sections 1.4 and 2.3).

An obvious bound is $2 \le \omega \le 3$, since the straightforward method of matrix multiplication uses $O(n^3)$ operations, on one hand, while on the other hand we need at least n^2 multiplications to compute n^2 independent matrix entries. The first known algorithm proving that $\omega < 3$ was Strassen's algorithm, detailed in Section 2.1, which starts with an algorithm for multiplying 2×2 matrices using seven multiplications and applies it recursively, giving $\omega \le \log_2 7$. This idea of exploiting recursion will be explored in the next section.

1.2 Asymptotic bilinear complexity via tensor ranks

The basic idea behind designing fast algorithms to multiply arbitrarily large matrices, thereby obtaining good upper bounds on ω, is to exploit recursion: multiplication of large matrices can be reduced to several smaller matrix multiplications. One obvious way to do this is to decompose

[3] Technically, division is not allowed, as the computation should be in $\mathbb{F}[X_{ij}, Y_{ij}]$, although this is no restriction if $\mathbb{F}$ is an infinite field (see [BCS97, Remark 15.2]).

the matrix into blocks, as in Strassen's original algorithm. Strassen's "laser method" [BCS97, Section 15.8] is a sophisticated version of this, where several matrix-multiplication tensors are efficiently packed into a single bilinear operation (not necessarily itself a matrix multiplication). The rank of the tensor—in fact the border rank, which will be defined below—is used to keep track of the complexity of the resulting recursive algorithm, and appears in the resulting inequality for ω. This idea of recursion is also behind the "group-theoretic" algorithms described in the next section.

We have mentioned that the exponent of matrix multiplication may be defined in terms of the rank $R\big(\langle n,n,n\rangle\big)$:

$$\omega(\mathbb{F}) = \inf\left\{\, \tau \in \mathbb{R} \;\middle|\; R\big(\langle n,n,n\rangle\big) = O(n^\tau) \,\right\}.$$

The reason for dealing with the rank rather than directly with the complexity measure is that the rank is better behaved with respect to certain operations, and this will be useful for deriving bounds on the asymptotic complexity via recursion. In particular [BCS97, Proposition 14.23], we have

$$R(\varphi_1 \otimes \varphi_2) \le R(\varphi_1) \otimes R(\varphi_2)$$

for bilinear maps φ_1 and φ_2, while the corresponding inequality with L in place of R is not known to be true. Let $\langle e,h,l\rangle$ be the tensor of $M_{e\times h} \times M_{h\times l} \to M_{e\times l}$ matrix multiplication. Since $\langle e,h,l\rangle \otimes \langle e',h',l'\rangle \cong \langle ee',hh',ll'\rangle$ [BCS97, Proposition 14.26], we have $R\big(\langle ee',hh',ll'\rangle\big) = R\big(\langle e,h,l\rangle\big)R\big(\langle e',h',l'\rangle\big)$. Using properties of the rank function, it is easy to derive bounds on ω given estimates of the rank of a particular tensor.

Example. If $R\big(\langle h,h,h\rangle\big) \le r$, then $h^\omega \le r$.

The first generalization is to allow rectangular matrices, via symmetrization: we have

$$R\big(\langle e,h,l\rangle\big) = R\big(\langle h,l,e\rangle\big) = R\big(\langle l,e,h\rangle\big)$$

(another nice property of the rank not shared by the multiplicative complexity), so if $R\big(\langle e,h,l\rangle\big) \le r$, then $R\big(\langle ehl,ehl,ehl\rangle\big) \le r^3$, and therefore

$$(ehl)^{\omega/3} \le r. \tag{4}$$

The next refinement is to multiply several matrices at once. But first we need to discuss border rank. The border rank appears as follows. The idea is that one may be able to approximate a tensor t of a certain rank by a family $t_1(\varepsilon) = \sum_{i=1}^r u_i(\varepsilon) \otimes v_i(\varepsilon) \otimes w_i(\varepsilon)$ of tensors of possibly smaller rank, meaning

$$\varepsilon^{1-q} t_1(\varepsilon) = t + O(\varepsilon)$$

for some positive integer q. The border rank $\underline{R}(t)$ is the smallest r for which this is possible. This has a geometric interpretation, studied by Landsberg [Lan08].

The border rank is always less than or equal to the rank, and shares some of its properties, including that of being hard to determine. Landsberg [Lan06] proved that $\underline{R}\big(\langle 2,2,2\rangle\big) = R\big(\langle 2,2,2\rangle\big) = 7$, but for $n = 3$ the best result known is $14 \le \underline{R}\big(\langle 3,3,3\rangle\big) \le 21$ (to be compared with the estimate $19 \le R\big(\langle 3,3,3\rangle\big) \le 23$ mentioned before).

The border rank may be strictly less than the rank. For instance, the rank of

$$t = x_1 \otimes y_1 \otimes (z_1 + z_2) + x_1 \otimes y_2 \otimes z_1 + x_2 \otimes y_1 \otimes z_1$$

is 3, but its border rank is only 2:

$$\varepsilon^{-1}t_1(\varepsilon) := \varepsilon^{-1}\left[(\varepsilon-1)x_1\otimes y_1\otimes z_1 + (x_1+\varepsilon x_2)\otimes(y_1+\varepsilon y_2)\otimes(z_1+\varepsilon z_2)\right] = t + O(\varepsilon),$$

as can be seen by expanding the left-hand side.

The importance of the border rank is that, as in this example, the original tensor may be recovered from $t_1(\varepsilon)$ by computing the coefficient of some power of ε; in other words, from such an approximate algorithm for computing t we may recover an exact one. This expansion increases the number of monomials, so this does not help to compute t itself; the magic happens when we compute $t^{\otimes N}$ for large N. Taking tensor powers corresponds to multiplying matrices recursively.

The border rank replaces the rank in a refinement of (4), so that $\underline{R}\big(\langle e,h,l\rangle\big) \le r$ implies $(ehl)^{\omega/3} \le r$. A bit of work, generalizing this to the case of several simultaneous matrix multiplications, results in Schönhage's *asymptotic sum inequality*

$$\underline{R}\left(\bigoplus_{i=1}^{s}\langle e_i,h_i,l_i\rangle\right) \le r \implies \sum_{i=1}^{s}(e_ih_il_i)^{\omega/3} \le r. \tag{5}$$

From these sorts of considerations, one can see that good bounds on the asymptotic complexity of matrix multiplication can be obtained by constructing specific tensors of small border rank which contain matrix tensors as components; this is the idea behind Strassen *et al.*'s laser method.

The principle of the laser method [BCS97, Proposition 15.41] is to look for a tensor t, of small border rank, which has a direct-sum decomposition into blocks each of which is isomorphic to a matrix tensor, and whose support is "tight", ensuring that in a large power of t one can find a sufficiently large direct sum of matrix tensors. Then one can apply (5).

This combinatorial method was used by Coppersmith and Winograd [CW90] to derive $\omega < 2.376$, the best estimate currently known.

1.3 Group-theoretic methods of fast matrix multiplication

As explained in the previous section, the general principle is to embed several simultaneous matrix multiplications in a single tensor, via some combinatorial construction to ensure that the embedding is efficient.

A rough sketch of Cohn *et al.*'s [CKSU05] "group-theoretic" algorithms is that they involve embedding matrix multiplication into multiplication in a group algebra $\mathbb{C}[G]$ of a finite group G. The embedding uses three subsets of G satisfying the "triple product property" to encode matrices as elements of the group algebra, so that the matrix product can be read off the corresponding product in $\mathbb{C}[G]$. The number of operations required to multiply two matrices is, therefore, less than or equal to the number of operations required to multiply two elements of $\mathbb{C}[G]$. As a ring, $\mathbb{C}[G] \cong M_{d_1\times d_1}(\mathbb{C}) \times \cdots \times M_{d_r\times d_r}(\mathbb{C})$, where $d_1,\ldots,d_r$ are the dimensions of the irreducible representations of G (see, for instance, [Lam01, Chapter 3]). This isomorphism may be realized as a Fourier transform on G, which can be computed efficiently. In other words, multiplication in $\mathbb{C}[G]$ is equivalent to several smaller matrix multiplications, and one can apply the algorithm recursively in order to get a bound on ω.

Cohn *et al.*'s embedding is of a very particular type, based on the following triple product property: if there are subsets X, Y, $Z \subseteq G$ such that $xx'^{-1}yy'^{-1}zz'^{-1} = 1$, then $x = x'$, $y = y'$, and $z = z'$. This realizes the $|X| \times |Y|$ by $|Y| \times |Z|$ matrix multiplication AB by sending a_{xy} to $\sum a_{xy}x^{-1}y$ and $b_{y'z}$ to $\sum b_{y'z}y'^{-1}z$; the triple product property ensures that one can extract

the matrix product from the product in the group algebra by looking at the coefficients of $x^{-1}z$ for $x \in X$ and $z \in Z$.

It may be more convenient, as in the previous section, to encode *several* matrix multiplications via the *simultaneous* triple product property: for X_i, Y_i, $Z_i \subseteq H$ one should have $x_i x_j'^{-1} y_j y_k'^{-1} z_k z_i'^{-1} = 1 \longrightarrow i = j = k$ and $x_i = x_i'$, $y_i = y_i'$, $z_i = z_i'$. It follows from (5) that

$$\sum_i \big(|X_i||Y_i||Z_i|\big)^{\omega/3} \leq \sum_k d_k^\omega.$$

We remark that the simultaneous triple product property in H reduces to the triple product property in the wreath product $G = H^n \rtimes \mathrm{Sym}_n$, so the groups actually output by this method turn out rather large.

From this initial description it is not at all clear what kinds of groups will give good bounds. To this end, Cohn *et al.* introduce several combinatorial constructions, analogous to those of Coppersmith and Winograd, which produce subsets satisfying the simultaneous triple product property inside powers H^k of a finite Abelian group H, and hence the triple product property inside wreath products of H with the symmetric group. This reproduces the known bounds $\omega < 2.376$, etc.

The group-theoretic method therefore provides another perspective on efficiently packing several independent matrix multiplications into one. In both cases the essential problem seems to be a combinatorial one, and one can state combinatorial conjectures which would imply $\omega = 2$.

1.4 Asymptotic complexity of other linear problems

One can also use recursive "divide-and-conquer" algorithms to prove that the asymptotic complexity of other problems in linear algebra is the same as that of matrix multiplication. This justifies the emphasis placed on matrix multiplication in numerical linear algebra.

As a simple example, we will begin with

Example (matrix inversion)**.** On one hand, we have the identity

$$\begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}^{-1} = \begin{pmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{pmatrix},$$

which shows that two $n \times n$ matrices may be multiplied by inverting a $3n \times 3n$ matrix. This shows that if an invertible $n \times n$ matrix can be inverted in $O(n^{\omega+\eta})$ operations, then the product of two arbitrary $n \times n$ matrices can also be computed in $O(n^{\omega+\eta})$ operations.

In the other direction, consider the identity

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}, \qquad S := D - CA^{-1}B.$$

This shows that inversion of $\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right) \in M_{2n\times 2n}(\mathbb{C})$ can be reduced to a certain (fixed) number of $n \times n$ matrix multiplications and inversions.[4] Unfortunately, the indicated inverses, e.g., A^{-1}, may not exist. This defect may be remedied by writing $\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right) = X = X^*(XX^*)^{-1}$. Now XX^* is a positive-definite Hermitian matrix, to which the indicated algorithm may be applied (both its upper-left block and its Schur complement will be positive-definite and Hermitian). We conclude that fast multiplication implies fast inversion of positive-definite Hermitian, and therefore of arbitrary (invertible), matrices.

[4] For instance, 2 inversions and 6 multiplications.

Example (LU decomposition)**.** Suppose, for instance, that one wishes to decompose a matrix A as $A = LUP$, where L is lower triangular and unipotent, U is upper triangular, and P is a permutation matrix. Note that not every matrix has such a decomposition; a sufficient condition for it to exist is that A have full row rank.

One can give a recursive algorithm [BCS97, Theorem 16.4], due to Bunch and Hopcroft, for computing the decomposition in case A has full row rank, via a 2×2 block decomposition of A. This involves one inversion of a triangular matrix, two applications of the algorithm to smaller matrices, and several matrix multiplications; we elide the details. Since multiplication and inversion can be done fast, analysis of this algorithm shows that if an $n \times n$ matrix can be multiplied in $O(n^{\omega+\eta})$ operations, then the LU decomposition of an $m \times n$ matrix can be done in $O(nm^{\omega+\eta-1})$ operations, that is $O(n^{\omega+\eta})$ in the case of a square matrix.

To show, conversely, that fast LU decomposition implies fast matrix multiplication, one notes that $\det A$ may be computed from an LU decomposition of A, and that computing determinants is at least as hard as matrix multiplication (cf. [BCS97, Theorem 16.7]). This shows that the exponents of matrix multiplication, LU decomposition, and determinants coincide.[5]

Further examples involving other linear problems may be found in the literature; see [BCS97] and also Section 2.3.

2 Numerical stability of linear problems

Numerical stability is just as important for the implementation of any algorithm as computational cost, since accumulation and propagation of roundoff errors may significantly distort the output of the algorithm, making the algorithm essentially useless. On the other hand, if roundoff error bounds can be established for a given algorithm, this guarantees that its output values can be trusted to lie within the regions provided by the error bounds. Moreover, such regions can typically be made small by increasing the hardware precision appropriately. Fast matrix multiplication algorithms, from Strassen's algorithm to the recent group-theoretic algorithms of Cohn *et al.*, can be analysed in a uniform fashion from the stability point of view [DDHK07].

The roundoff-error analysis of Strassen's method was first performed by Brent ([Bre70, Hig90], see also [Hig02, chap. 23]). The analysis of subsequent Strassen-like algorithms is due a number of authors, most notably by Bini and Lotti [BL80]. This latter approach was advanced in [DDHK07] to build an inclusive framework that accommodates all Strassen-like algorithms based on stationary partitioning, bilinear algorithms with non-stationary partitioning, and finally the group-theoretic algorithms of the kind developed in [CU03] and [CKSU05]. Moreover, combining this framework with a result of Raz [Raz03], one can prove that there exist numerically stable matrix multiplication algorithms which perform $O(n^{\omega+\eta})$ operations, for arbitrarily small $\eta > 0$, where ω is the exponent of matrix multiplication.

The starting point of the error analysis [DDHK07] is the so-called *classical model of rounded arithmetic,* where each arithmetic operation introduces a small multiplicative error, i.e., the computed value of each arithmetic operation $\mathrm{op}(a, b)$ is given by $\mathrm{op}(a, b)(1 + \theta)$ where $|\theta|$ is bounded by some fixed *machine precision* ε but is otherwise arbitrary. The arithmetic operations in classical arithmetic are $\{+, -, \cdot\}$. The roundoff errors are assumed to be introduced by *every execution* of any arithmetic operation. It is further assumed that all algorithms output the exact value in the absence of roundoff errors (i.e., when all errors θ are zero).

[5] Compare this result on determinants with the problem of computing the permanent, which is NP-hard!

The error analysis can be performed with respect to various norms on the matrices A, B, $C = AB$, as will be made clear in the next section. It leads to error bounds of the form

$$\|C_{\text{comp}} - C\| \leq \mu(n)\varepsilon\|A\|\,\|B\| + O(\varepsilon^2), \tag{6}$$

with $\mu(n)$ typically low-degree polynomials in the order n of the matrices involved, so that $\mu(n) = O(n^c)$ for some constant c. Switching from one norm to another is always possible, using the equivalence of norms on a finite-dimensional space, but this may incur additional factors that depend on n.

2.1 Recursive matrix multiplication: Strassen and beyond

In his breakthrough paper [Str69], Strassen observed that the multiplication of two 2×2 block matrices requires only 7 (instead of 8) block multiplications, and used that remarkable observation recursively to obtain a matrix-multiplication algorithm with running time $O(n^{\log_2 7})$. Precisely, the product

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

can be computed by calculating the submatrices

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22})B_{11} \\ M_3 &= A_{11}(B_{12} - B_{22}) \\ M_4 &= A_{22}(B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12})B_{22} \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

and then combining them linearly as

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6. \end{aligned}$$

Starting with matrices of dyadic order, this algorithm can be applied by recursively partitioning each matrix into four square blocks and running these computations. This yields running time $O(n^{\log_2 7}) \approx O(n^{2.81})$. Since any matrix can be padded with zeros to achieve the nearest dyadic order, the dyadic size assumption is not restrictive at all.

The breakthrough of Strassen generated a flurry of activity in the area, leading to a number of subsequent improvements, among those by Pan [Pan78], Bini *et al.* [BCRL79], Schönhage [Sch81], Strassen [Str87], and eventually Coppersmith and Winograd [CW90]. Each of these algorithms is Strassen-like, i.e., uses recursive partitioning and a special "trick" to reduce the number of block matrix multiplications.

Such recursive algorithms for matrix multiplication can be analysed as follows. Recall that bilinear functions can be evaluated via bilinear algorithms, as in Equation (2). Since they do not

use commutativity of the coordinates, these algorithms apply equally well when the input entries are elements of a non-commutative algebra; their recursive use for matrix multiplication is then straightforward. A *bilinear non-commutative algorithm* (see [BL80] or [BD78]) that computes products $C = AB$ in $M_{k\times k}(\mathbb{F})$ using t non-scalar multiplications over a subfield $\mathbb{H} \subseteq \mathbb{F}$ (not necessarily equal to $\mathbb{F}$)[6] is determined by three $k^2 \times t$ matrices U, V and W with elements in $\mathbb{H}$ such that

$$c_{hl} = \sum_{s=1}^{t} w_{rs} P_s, \quad \text{where } P_s = \left(\sum_{i=1}^{k^2} u_{is} x_i\right)\left(\sum_{j=1}^{k^2} v_{js} y_j\right), \qquad \begin{array}{l} r = k(h-1)+l, \\ h,l = 1,\ldots,k, \end{array} \tag{7}$$

where x_i (resp. y_j) are the elements of $A = (a_{ij})$ (resp. of $B = (b_{ij})$) ordered column-wise, and $C = (c_{ij})$ is the product $C = AB$.

For an arbitrary n, the algorithm consists of recursive partitioning and applying (7) to compute products of resulting block matrices. More precisely, suppose that A and B are of size $n\times n$, where n is a power of k (which can always be achieved by padding the matrices A and B with zero columns and rows, as we already mentioned). Partition A and B into k^2 square blocks A_{ij}, B_{ij} of size $(n/k)\times(n/k)$. Then the blocks C_{hl} of the product $C = AB$ can be computed by applying (7) to the blocks of A and B, where each block A_{ij}, B_{ij} has to be again partitioned into k^2 square sub-blocks to compute the t products P_s and then the blocks C_{hl}. The algorithm obtained by running this recursive procedure $\log_k n$ times computes the product $C = AB$ using at most $O(n^{\log_k t})$ multiplications.

Theorem ([DDHK07, Theorem 3.1])**.** A bilinear non-commutative algorithm for matrix multiplication based on stationary partitioning is stable. It satisfies the error bound (6) where $\|\cdot\|$ is the maximum-entry norm and where

$$\mu(n) = \left(1 + \max_{r,s}(\alpha_s + \beta_s + \gamma_r + 3)\log_k n\right) \cdot \left(\mathrm{emax} \cdot \|U\|\,\|V\|\,\|W\|\right)^{\log_k n}.$$

Here $\alpha_s = \lceil \log_2 a_s \rceil$, $\beta_s = \lceil \log_2 b_s \rceil$ and $\gamma_r = \lceil \log_2 c_r \rceil$ where a_s and b_s (resp. c_r) are the number of non-zero entries of U and V (resp. W) in column s (resp. row r), while emax is an integer that depends (in a rather involved way) on the sparsity pattern of the matrices U, V and W.

This theorem can be subsequently combined with the result of Raz [Raz03] that the exponent of matrix multiplication is achieved by bilinear non-commutative algorithms [Raz03] to produce an important corollary:

Corollary ([DDHK07, Theorem 3.3])**.** For every $\eta > 0$ there exists an algorithm for multiplying n-by-n matrices which performs $O(n^{\omega+\eta})$ operations (where ω is the exponent of matrix multiplication) and which is numerically stable, in the sense that it satisfies the error bound (6) with $\mu(n) = O(n^c)$ for some constant c depending on η but not n.

The analysis of stationary algorithms extends to bilinear matrix multiplication algorithms based on non-stationary partitioning. This means that the matrices $A^{[j]}_{s,\mathrm{comp}}$ and $B^{[j]}_{s,\mathrm{comp}}$ are partitioned into $k\times k$ square blocks, but k depends on the level of recursion, i.e., $k = k(j)$, and the corresponding matrices U, V and W also depend on j: $U = U(j)$, $V = V(j)$, $W = W(j)$. Otherwise the algorithm proceeds exactly like the stationary algorithms.

Finally, algorithms that combine recursive non-stationary partitioning with pre- and post-processing given by linear maps $\textsc{Pre}_n()$ and $\textsc{Post}_n()$ acting on matrices of an arbitrary order n

[6] Field extensions have no effect on the asymptotic complexity, but changing $\mathbb{H}$ will affect the constants in $\mu(n)$.

can be analysed using essentially the same approach [DDHK07]. Suppose that the matrices A and B are each (linearly) pre-processed, then partitioned into blocks, respective pairs of blocks are multiplied recursively and assembled into a large matrix, which is then (linearly) post-processed to obtain the resulting matrix C.

The analysis in [DDHK07] is performed for an arbitrary *consistent* (i.e., submultiplicative) norm $\|\cdot\|$ that in addition must be defined for matrices of all sizes and must satisfy the condition

$$\max_s \|M_s\| \leq \|M\| \leq \sum_s \|M_s\| \tag{8}$$

whenever the matrix M is partitioned into blocks $(M_s)_s$ (an example of such a norm is provided by the 2-norm $\|\cdot\|_2$). Note that the previously mentioned maximum-entry norm satisfies (8) but is not consistent, i.e., does not satisfy

$$\|AB\| \leq \|A\| \cdot \|B\| \qquad \text{for all } A,\ B.$$

Denoting the norms of pre- and post- processing maps subordinate to the norm $\|\cdot\|$ by $\|\cdot\|_{\mathrm{op}}$, we suppose that the pre- and post-processing is performed with errors

$$\begin{aligned} \|\textsc{Pre}_n(M)_{\mathrm{comp}} - \textsc{Pre}_n(M)\|_{\mathrm{op}} &\leq f_{\mathrm{pre}}(n)\varepsilon\|M\| + O(\varepsilon^2), \\ \|\textsc{Post}_n(M)_{\mathrm{comp}} - \textsc{Post}_n(M)\|_{\mathrm{op}} &\leq f_{\mathrm{post}}(n)\varepsilon\|M\| + O(\varepsilon^2), \end{aligned}$$

where n is the order of the matrix M. As before, we denote by $\mu(n)$ the coefficient of ε in the final error bound (6).

Under all these assumptions, the following error estimate follows:

Theorem ([DDHK07, Theorem 3.5])**.** A recursive matrix multiplication algorithm based on non-stationary partitioning with pre- and post-processing is stable. It satisfies the error bound (6), with the function μ satisfying the recursion

$$\mu(n_j) = \mu(n_{j+1})t_j\|\textsc{Post}_{n_j}\|_{\mathrm{op}}\|\textsc{Pre}_{n_j}\|_{\mathrm{op}}^2 + 2f_{\mathrm{pre}}(n_j)t_j\|\textsc{Post}_{n_j}\|_{\mathrm{op}} + f_{\mathrm{post}}(n_j)\|\textsc{Pre}_{n_j}\|_{\mathrm{op}}^2$$

for $j = 1, \ldots, p$.

2.2 Group-theoretic matrix multiplication

In this section we describe the group-theoretic constructions of Cohn *et al.* Our exposition closely follows the pertinent parts of [DDHK07]. To give a general idea about group-theoretic fast matrix multiplication, we must first recall some basic definitions from algebra.

Definition (semidirect product)**.** If H is any group and Q is a group which acts (on the left) by automorphisms of H, with $q \cdot h$ denoting the action of $q \in Q$ on $h \in H$, then the *semidirect product* $H \rtimes Q$ is the set of ordered pairs (h, q) with the multiplication law

$$(h_1, q_1)(h_2, q_2) = (h_1(q_1 \cdot h_2), q_1q_2). \tag{9}$$

We will identify $H \times \{1_Q\}$ with H and $\{1_H\} \times Q$ with Q, so that an element $(h, q) \in H \rtimes Q$ may also be denoted simply by hq. Note that the multiplication law of $H \rtimes Q$ implies the relation $qh = (q \cdot h)q$.

Definition (wreath product)**.** If H is any group, S is any finite set, and Q is a group with a left action on S, the *wreath product* $H \wr Q$ is the semidirect product $(H^S) \rtimes Q$ where Q acts on the direct product of $|S|$ copies of H by permuting the coordinates according to the action of Q on S. (To be more precise about the action of Q on H^S, if an element $h \in H^S$ is represented as a function $h\colon S \to H$, then $q \cdot h$ represents the function $s \mapsto h(q^{-1}(s))$.)

Definition (triple product property, simultaneous triple product property)**.** If H is a group and X, Y, Z are three subsets, we say X, Y, Z satisfy the *triple product property* if it is the case that for all $q_x \in Q(X)$, $q_y \in Q(Y)$, $q_z \in Q(Z)$, if $q_x q_y q_z = 1$ then $q_x = q_y = q_z = 1$. Here $Q(X) = Q(X, X)$ is the set of quotients; $Q(S, T) := \left\{ st^{-1} \;\middle|\; s \in S,\ t \in T \right\} \subseteq H$.

If $\{ (X_i, Y_i, Z_i) \mid i \in I \}$ is a collection of ordered triples of subsets of H, we say that this collection satisfies the *simultaneous triple product property (STPP)* if it is the case that for all i, j, $k \in I$ and all $q_x \in Q(X_i, X_j)$, $q_y \in Q(Y_j, Y_k)$, $q_z \in Q(Z_k, Z_i)$, if $q_x q_y q_z = 1$ then $q_x = q_y = q_z = 1$ and $i = j = k$.

Definition (Abelian STP family)**.** An *Abelian STP family* with growth parameters (α, β) is a collection of ordered triples (H_N, Υ_N, k_N), defined for all $N > 0$, satisfying

1. H_N is an Abelian group.
2. $\Upsilon_N = \{ (X_i, Y_i, Z_i) \mid i = 1, 2, \ldots, N \}$ is a collection of N ordered triples of subsets of H_N satisfying the simultaneous triple product property.
3. $|H_N| = N^{\alpha + o(1)}$.
4. $k_N = \prod_{i=1}^{N} |X_i| = \prod_{i=1}^{N} |Y_i| = \prod_{i=1}^{N} |Z_i| = N^{\beta N + o(N)}$.

Recall from Section 1.3 that in [CKSU05] matrix-multiplication algorithms are constructed based on families of wreath products of Abelian groups.

To get into more details, we must recall basic facts about the discrete Fourier transform of an Abelian group. For an Abelian group H, let $\widehat{H}$ denote the set of all homomorphisms from H to S^1, the multiplicative group of complex numbers with unit modulus. Elements of $\widehat{H}$ are called *characters* and are usually denoted by the letter χ. The sets H, $\widehat{H}$ have the same cardinality. When H_1, H_2 are two Abelian groups, there is a canonical bijection between the sets $\widehat{H_1} \times \widehat{H_2}$ and $(H_1 \times H_2)^{\wedge}$; this bijection maps an ordered pair (χ_1, χ_2) to the character χ given by the formula $\chi(h_1, h_2) = \chi_1(h_1)\chi_2(h_2)$. Just as the symmetric group Sym_n acts on H^n via the formula $\sigma \cdot (h_1, h_2, \ldots, h_n) = (h_{\sigma^{-1}(1)}, h_{\sigma^{-1}(2)}, \ldots, h_{\sigma^{-1}(n)})$, there is a left action of Sym_n on the set $\widehat{H}^n$ defined by the formula $\sigma \cdot (\chi_1, \chi_2, \ldots, \chi_n) = (\chi_{\sigma^{-1}(1)}, \chi_{\sigma^{-1}(2)}, \ldots, \chi_{\sigma^{-1}(n)})$.

Notation. The notation $\Xi(H^n)$ will be used to denote a subset of $\widehat{H}^n$ containing exactly one representative of each orbit of the Sym_n action on $\widehat{H}^n$. An orbit of this action is uniquely determined by a multiset consisting of n characters of H, so the cardinality of $\Xi(H^n)$ is equal to the number of such multisets, i.e. $\binom{|H|+N-1}{N}$.

Given an Abelian STP family, the corresponding recursive matrix multiplication algorithm is defined as follows. Given a pair of n-by-n matrices A, B, find the minimum N such that $k_N \cdot N! \geq n$, and denote the group H_N by H. If $N! \geq n$, multiply the matrices using an arbitrary algorithm. (This is the base of the recursion.) Otherwise reduce the problem of computing the matrix product AB to $\binom{|H|+N-1}{N}$ instances of $N! \times N!$ matrix multiplication, using a reduction based on the discrete Fourier transform of the Abelian group H^N.

Padding the matrices with additional rows and columns of 0's if necessary, one may assume that $k_N \cdot N! = n$. Define subsets $X, Y, Z \subseteq H \wr \mathrm{Sym}_N$ as

$$X = \left(\prod_{i=1}^{N} X_i\right) \times \mathrm{Sym}_N, \qquad Y = \left(\prod_{i=1}^{N} Y_i\right) \times \mathrm{Sym}_N, \qquad Z = \left(\prod_{i=1}^{N} Z_i\right) \times \mathrm{Sym}_N .$$

These subsets satisfy the triple product property [CKSU05]. Note that $|X| = |Y| = |Z| = n$. Now treat the rows and columns of A as being indexed by the sets X, Y, respectively; treat the rows and columns of B as being indexed by the sets Y, Z, respectively.

The algorithm uses two auxiliary vector spaces $\mathbb{C}[H \wr \mathrm{Sym}_N]$ and $\mathbb{C}[\widehat{H}^N \rtimes \mathrm{Sym}_N]$, each of dimensionality $|H|^N N!$ and each with a specific basis: the basis for $\mathbb{C}[H \wr \mathrm{Sym}_N]$ is denoted by $\{ \mathbf{e}_g \mid g \in H \wr \mathrm{Sym}_N \}$, and the basis for $\mathbb{C}[\widehat{H}^N \rtimes \mathrm{Sym}_N]$ is denoted by $\{ \mathbf{e}_{\chi,\sigma} \mid \chi \in \widehat{H}^N,\ \sigma \in \mathrm{Sym}_N \}$.

The Abelian STP algorithm from [CKSU05] performs the following steps, which will be labelled according to whether they perform arithmetic or not. (For example, a permutation of the components of a vector does not involve any arithmetic.)

1. **Embedding** (NO ARITHMETIC): Compute the following pair of vectors in $\mathbb{C}[H \wr \mathrm{Sym}_N]$.

$$\begin{aligned} a &:= \sum_{x \in X} \sum_{y \in Y} A_{xy} \mathbf{e}_{x^{-1}y} \\ b &:= \sum_{y \in Y} \sum_{z \in Z} B_{yz} \mathbf{e}_{y^{-1}z}. \end{aligned}$$

2. **Fourier transform** (ARITHMETIC): Compute the following pair of vectors in $\mathbb{C}[\widehat{H}^N \rtimes \mathrm{Sym}_N]$.

$$\begin{aligned} \hat{a} &:= \sum_{\chi \in \widehat{H}^N} \sum_{\sigma \in \mathrm{Sym}_N} \left(\sum_{h \in H^N} \chi(h) a_{\sigma h} \right) \mathbf{e}_{\chi,\sigma}. \\ \hat{b} &:= \sum_{\chi \in \widehat{H}^N} \sum_{\sigma \in \mathrm{Sym}_N} \left(\sum_{h \in H^N} \chi(h) b_{\sigma h} \right) \mathbf{e}_{\chi,\sigma}. \end{aligned}$$

3. **Assemble matrices** (NO ARITHMETIC): For every $\chi \in \Xi(H^N)$, compute the following pair of matrices A^χ, B^χ, whose rows and columns are indexed by elements of Sym_N.

$$\begin{aligned} A^\chi_{\rho\sigma} &:= \hat{a}_{\rho \cdot \chi, \sigma\rho^{-1}} \\ B^\chi_{\sigma\tau} &:= \hat{b}_{\sigma \cdot \chi, \tau\sigma^{-1}} \end{aligned}$$

4. **Multiply matrices** (ARITHMETIC): For every $\chi \in \Xi(H^N)$, compute the matrix product $C^\chi := A^\chi B^\chi$ by recursively applying the Abelian STP algorithm.

5. **Disassemble matrices** (NO ARITHMETIC): Compute a vector $\hat{c} := \sum_{\chi,\sigma} \hat{c}_{\chi,\sigma} \mathbf{e}_{\chi,\sigma} \in \mathbb{C}[\widehat{H}^N \rtimes \mathrm{Sym}_N]$ whose components $\hat{c}_{\chi,\sigma}$ are defined as follows. Given χ, σ, let $\chi_0 \in \Xi(H^N)$ and $\tau \in \mathrm{Sym}_N$ be such that $\chi = \tau \cdot \chi_0$. Let

$$\hat{c}_{\chi,\sigma} := C^{\chi_0}_{\tau,\sigma\tau}.$$

6. **Inverse Fourier transform** (ARITHMETIC): Compute the following vector $c \in \mathbb{C}[H \wr \mathrm{Sym}_N]$.

$$c := \sum_{h \in H^N} \sum_{\sigma \in \mathrm{Sym}_N} \left(\frac{1}{|H|^N} \sum_{\chi \in \widehat{H}^N} \chi(-h) \hat{c}_{\chi,\sigma} \right) \mathbf{e}_{\sigma h}.$$

7. **Output** (NO ARITHMETIC): Output the matrix $C = (C_{xz})$ whose entries are given by the formula

$$C_{xz} := c_{x^{-1}z}.$$

The main result of [DDHK07] establishes the numerical stability of all Abelian STP algorithms.

Theorem ([DDHK07, Theorem 4.13])**.** If $\{ (H_N, \Upsilon_N, k_N) \}$ is an Abelian STP family with growth parameters (α, β), then the corresponding Abelian STP algorithm is stable. It satisfies the error bound (6), with the Frobenius norm and the function μ of order

$$\mu(n) = n^{\frac{\alpha+2}{2\beta} + o(1)}.$$

Remark ([DDHK07, Remark 4.15])**.** The running time of an Abelian STP algorithm can also be bounded in terms of the growth parameters of the Abelian STP family. Specifically, the running time is [CKSU05] $O\left(n^{(\alpha-1)/\beta + o(1)}\right)$. Note the curious interplay between the two exponents, $(\alpha-1)/\beta$ and $(\alpha+2)/2\beta$: their sum is always bigger than 3, since $\alpha \geq 2\beta + 1$ is one of the requirements for an Abelian STP construction:

$$\frac{\alpha-1}{\beta} + \frac{\alpha+2}{2\beta} = \frac{3\alpha}{2\beta} \geq \frac{6\beta+3}{2\beta} > 3.$$

2.3 Matrix decompositions and other linear problems

The results about matrix multiplication from the previous section can be extended to show that essentially *all* linear algebra operations can also be done stably, in time $O(n^\omega)$ or $O(n^{\omega+\eta})$, for arbitrary $\eta > 0$ [DDH07]. For simplicity, whenever an exponent contains "$+\eta$", it will henceforth mean "for any $\eta > 0$". Below we summarize the main results of [DDH07].

The first result in [DDH07] can be roughly summarized by saying that n-by-n matrices can be multiplied in $O(n^{\omega+\eta})$ operations *if and only if* n-by-n matrices can be inverted stably in $O(n^{\omega+\eta})$ operations. Some extra precision is necessary to make this claim; the cost of extra precision is included in the $O(n^\eta)$ factor.

Other results in [DDH07] may be summarized by saying that if n-by-n matrices can be multiplied in $O(n^{\omega+\eta})$ *arithmetic* operations, then the QR decomposition can be computed stably (moreover, linear systems and least squares problems can be solved stably) in $O(n^{\omega+\eta})$ *arithmetic* operations. These results do not require extra precision, which is why one needs to count arithmetic operations rather than bit operations.

The QR decomposition can be used to stably compute a rank-revealing decomposition, the (generalized) Schur form, and the singular value decomposition, all in $O(n^{\omega+\eta})$ *arithmetic* operations. Computing (generalized) eigenvectors from the Schur form, can be done by solving the (generalized) Sylvester equation, all of which can be done stably in $O(n^{\omega+\eta})$ *bit* operations.

Here are a few more details about the work in [DDH07]. The paper starts off by reviewing conventional block algorithms used in libraries like LAPACK [ABB+99] and ScaLAPACK [BCC+97]. The normwise backward stability of these algorithms was shown earlier [Hig90, DHS95, Hig02] using (6) as an assumption. This means that these algorithms are

guaranteed to produce the exact answer (e.g., solution of a linear system) for a matrix $\hat{C}$ close to the actual input matrix C, where close means close in norm:

$$\|\hat{C} - C\| = O(\varepsilon)\|C\|.$$

Here the $O(\varepsilon)$ is interpreted to include a factor n^c for a modest constant c.

The running-time analysis of these block algorithms in [DDH07] shows that these block algorithms run only as fast as $O(n^{\frac{9-2\gamma}{4-\gamma}})$ operations, where $O(n^\gamma)$ is the operation count of matrix multiplication, with γ used instead of $\omega + \eta$ to simplify notation. Even if γ were to drop from 3 to 2, the exponent $\frac{9-2\gamma}{4-\gamma}$ would only drop from 3 to 2.5, providing only a partial improvement. However, further results in [DDH07] demonstrate that one can do better.

The next step in [DDH07] is the application of known divide-and-conquer algorithms for reducing the complexity of matrix inversion to the complexity of matrix multiplication. These algorithms are not backward stable in the conventional sense. However, they can be shown to achieve the same forward error bound (bound on the norm of the error in the output) as a conventional backward stable algorithm, provided that they use just $O(\log^p n)$ times as many bits of precision in each arithmetic operation (for some $p > 0$) as a conventional algorithm. Such algorithms are called *logarithmically stable.*

Incorporating the cost of this extra precise arithmetic in the analysis only increases the total cost by a factor at most $\log^{2p} n$. Therefore, if there are matrix multiplication algorithms running in $O(n^{\omega+\eta})$ operations for any $\eta > 0$, then these logarithmically stable algorithms for operations like matrix inversion also run in $O(n^{\omega+\eta})$ operations for any $\eta > 0$, and satisfy the same error bound as a conventional algorithm.

A divide-and-conquer algorithm for QR decomposition from [EG00] is simultaneously backward stable in the conventional normwise sense (i.e., without extra precision), and runs in $O(n^{\omega+\eta})$ operations for any $\eta > 0$. This algorithm may be in turn used to solve linear systems, least-squares problems, and compute determinants equally stably and fast. The same idea applies to LU decomposition but stability depends on a particular pivoting assumption [DDH07].

The QR decomposition can then be used to compute a rank-revealing URV decomposition of a matrix A. This means that U and V are orthogonal, R is upper triangular, and R reveals the rank of A in the following sense: Suppose $\sigma_1 \geq \cdots \geq \sigma_n$ are the singular values of A. Then for each r, $\sigma_{\min}(R(1:r, 1:r))$ is an approximation of σ_r and $\sigma_{\max}(R(r+1:n, r+1:n))$ is an approximation of σ_{r+1}. The algorithm in [DDH07] is *randomized,* in the sense that the approximations of σ_r and σ_{r+1} are reasonably accurate with high probability.

Finally, the QR and URV decompositions in algorithms for the (generalized) Schur form of nonsymmetric matrices (or pencils) [BDG97] lower their complexity to $O(n^{\omega+\eta})$ arithmetic operations while maintaining normwise backward stability. The singular-value decomposition may in turn be reduced to solving an eigenvalue problem with the same complexity. Computing (generalized) eigenvectors can only be done in a logarithmically stable way from the (generalized) Schur form. This is done by providing a logarithmically stable algorithm for solving the (generalized) Sylvester equation, and using this to compute eigenvectors.

This covers nearly all standard dense linear algebra operations (LU decomposition, QR decomposition, matrix inversion, linear equation solving, solving least squares problems, computing the (generalized) Schur form, computing the SVD, and solving (generalized) Sylvester equations) and shows that all those problems can be solved stably and asymptotically as fast as the fastest matrix multiplication algorithm that may ever exist (whether the fastest matrix multiplication algorithm is stable or not). For all but matrix inversion and solving (generalized) Sylvester equations, stability means backward stability in a normwise sense, and the complexity is measured by the usual count of arithmetic operations.

References

[ABB+99] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Blackford, and D. Sorensen. *LAPACK Users' Guide (third edition)*. SIAM, Philadelphia, 1999.

[BCC+97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, 1997.

[BCRL79] D. Bini, M. Capovani, F. Romani, and G. Lotti. $O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. *Inform. Process. Lett.*, 8(5):234–235, 1979.

[BCS97] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1997. With the collaboration of Thomas Lickteig.

[BD78] R. W. Brockett and D. Dobkin. On the optimal evaluation of a set of bilinear forms. *Linear Algebra and Appl.*, 19(3):207–235, 1978.

[BDG97] Z. Bai, J. Demmel, and M. Gu. Inverse free parallel spectral divide and conquer algorithms for nonsymmetric eigenproblems. *Num. Math.*, 76:279–308, 1997. UC Berkeley CS Division Report UCB//CSD-94-793, Feb 94.

[BL80] D. Bini and D. Lotti. Stability of fast algorithms for matrix multiplication. *Num. Math.*, 36:63–72, 1980.

[Bre70] R. P. Brent. Algorithms for matrix multiplication. Computer Science Dept. Report CS 157, Stanford University, 1970.

[CKSU05] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. In *Foundations of Computer Science. 46th Annual IEEE Symposium on 23–25 Oct 2005*, pages 379–388. 2005.

[CU03] H. Cohn and C. Umans. A group-theoretic approach to matrix multiplication. In *Foundations of Computer Science. 44th Annual IEEE Symposium*, pages 438–449. 2003.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990.

[DDH07] J. Demmel, I. Dumitriu, and O. Holtz. Fast linear algebra is stable. *Numer. Math.*, 108(1):59–91, 2007.

[DDHK07] J. Demmel, I. Dumitriu, O. Holtz, and R. Kleinberg. Fast matrix multiplication is stable. *Numer. Math.*, 106(2):199–224, 2007.

[DHS95] J. Demmel, N. J. Higham, and R. Schreiber. Stability of block LU factorization, 1995.

[EG00] E. Elmroth and F. Gustavson. Applying recursion to serial and parallel qr factorization. *SIAM J. Comput.*, 44(4):605–624, 2000.

[Hig90] N. J. Higham. Exploiting fast matrix multiplication within the Level 3 BLAS. *ACM Trans. Math. Soft.*, 16:352–368, 1990.

[Hig02] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 2nd edition, 2002.

[Lam01] T. Y. Lam. *A first course in noncommutative rings*, volume 131 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 2001.

[Lan06] J. M. Landsberg. The border rank of the multiplication of 2×2 matrices is seven. *J. Amer. Math. Soc.*, 19(2):447–459 (electronic), 2006.

[Lan08] J. M. Landsberg. Geometry and the complexity of matrix multiplication. *Bull. Amer. Math. Soc. (N.S.)*, 45(2):247–284, 2008.

[Pan78] V. Ya. Pan. Strassen's algorithm is not optimal. Trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich., 1978)*, pages 166–176. IEEE, Long Beach, Calif., 1978.

[Raz03] R. Raz. On the complexity of matrix product. *SIAM J. Comput.*, 32(5):1356–1369 (electronic), 2003.

[Sch81] A. Schönhage. Partial and total matrix multiplication. *SIAM J. Comput.*, 10(3):434–455, 1981.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, August 1969.

[Str87] V. Strassen. Relative bilinear complexity and matrix multiplication. *J. Reine Angew. Math.*, 375/376:406–443, 1987.

[Win71] S. Winograd. On multiplication of 2×2 matrices. *Linear Algebra and Appl.*, 4:381–388, 1971.